A bounded-search iterated greedy algorithm for the distributed permutation flowshop scheduling problem*

Victor Fernandez-Viagas[1]†, Jose M. Framinan[1]

[1] Industrial Management, School of Engineering, University of Seville,

Ave. Descubrimientos s/n, E41092 Seville, Spain, {vfernandezviagas,framinan}@us.es

May 21, 2015

Abstract

As the interest of practitioners and researchers in scheduling in a multi-factory environment is growing, there is an increasing need to provide efficient algorithms for this type of decision problems, characterised by simultaneously addressing the assignment of jobs to different factories/workshops and their subsequent scheduling. Here we address the so-called distributed permutation flowshop scheduling problem, in which a set of jobs has to be scheduled over a number of identical factories, each one with its machines arranged as a flowshop. Several heuristics have been designed for this problem, although there is no direct comparison among them. In this paper, we propose a new heuristic which exploits the specific structure of the problem. The computational experience carried out on a well-known testbed shows that the proposed heuristic outperforms existing state-of-the-art heuristics, being able to obtain better upper bounds for more than one quarter of the problems in the testbed.

Keywords: distributed permutation flowshop, scheduling, heuristics, iterated greedy algorithm.

*Preprint submitted to International Journal of Production Research, http://dx.doi.org/10.1080/00207543.2014.948578.

†Corresponding author. Tel.: +34-954487220.

1 Introduction

Although the majority of scheduling problems in the literature assumes that the jobs have to be scheduled in a single factory, the number of companies using this environment is decreasing (Moon et al., 2002; Naderi and Ruiz, 2010; Wang et al., 2013). Instead, a multi-factory environment is becoming more and more important, since it reduces production costs and risks while increases the product quality (see e.g. Kahn et al., 2004). As a consequence, distributed production scheduling problems dealing with both the allocation of jobs to different factories and their subsequent scheduling has been receiving an increasing attention in the literature (e.g. Jia et al., 2003, 2007; Chan et al., 2005; Chan and Chan, 2010; Chung et al., 2009). Among this type of decision problems, Naderi and Ruiz (2010) recently presented the so-called distributed permutation flowshop scheduling problem, or DPFSP in the following. In this problem, a set of n jobs has to be processed in one of F identical factories, each one consisting of m machines that all jobs must visit in the same order. The decisions involved in this problem are to simultaneously decide in which factory the jobs have to be processed and which is the sequence of the jobs for each factory. If the objective sought for this problem is the minimisation of the global makespan (i.e. the maximum makespan across the F factories), then the problem can be denoted as $DF|prmu|C_{max}$ (see Naderi and Ruiz, 2010) following the well-known notation of Graham et al. (1979).

The problem under consideration can be seen as a generalization of the well-known Permutation Flowshop Scheduling Problem (PFSP) since, once a set of jobs has been assigned to a factory, the remaining decision problem is a PFSP. Since the latter problem is known to be NP-hard for more than two machines (Garey et al., 1976), $DF|prmu|C_{max}$ is also NP-hard for $m > 2$. Therefore, researchers have focused on finding methods able to find good –but not necessarily optimal– solutions within a reasonable computational effort. Among them, the works by Naderi and Ruiz (2010); Gao and Chen (2011); Gao et al. (2013); Wang et al. (2013); Lin et al. (2013) have provided increasingly better heuristics for the problem.

In this paper, a new efficient approximate algorithm is proposed for the DPFSP. This algorithm is based on one of the most efficient methods for the PFSP (i.e. the iterated greedy algorithm) and exploits the specific structure of solutions of the problem, thus allowing an up-to

30% reduction of the search space. Furthermore, two new efficient local search methods are embedded in our proposal to improve the so-found solutions. The results prove that the proposed algorithm is very effective, being the best one for each size of the problems in the testbed of Naderi and Ruiz (2010). Indeed, new upper bounds are found for 27.6% of the instances in this testbed.

The remainder of the paper is organized as follows: in Section 2 the problem under consideration is described along with its state-of-the-art; in Section 3 the proposed algorithm is detailed; in Section 4 the algorithm is compared with the rest of existing heuristics in the literature; and, finally, in Section 5 the main conclusions are presented.

2 Problem statement and state-of-the-art

The problem under consideration can be stated as follows: n jobs have to be scheduled in one of the F flowshop-factories consisting of m machines. Each factory is identical with the same set of m machines and is able to process all jobs. Once a job is assigned to a factory, it has to be processed there without being transferred to another factory. On each machine i, each job j has a processing time denoted as p_{ij} regardless the factory f where the job is processed. The problem determines the sequence π^f, formed by n_f jobs, to be scheduled in each factory f. Therefore, a solution π is formed by the sequence in each factory ($\pi = [\pi^1, \ldots, \pi^f, \ldots, \pi^F]$). Let us $C^f_{i,j}$ be the completion time of job j in machine i when assigned to factory f, and $C^f_{max} = C_{max}(\pi^f)$ the makespan of factory f. Then $C_{max} = C_{max}(\pi)$ denotes the global makespan. i.e. the completion time of the last job to be processed in any factory. Additionally, $\pi^f[i]$ is employed to denote the element of factory f in position i. By using f_{max} to denote the factory with maximum makespan, the global makespan can be also written as $C^{f_{max}}_{max}$.

On one hand, the DPFSP can be seen as a special case of the distributed assembly permutation flowshop scheduling problem (DAPFSP), see Hatami et al. (2013). When each factory is formed by exactly two machines, the problem has been also studied in the literature under the name of *Parallel Flowline* (Vairaktarakis and Elhafsi, 2000) or *Parallel Flowshops* (Cao and Chen, 2003). In this special case, the problem turns to be a pure assignment problem, since Johnson's rule

(Johnson, 1954) can be applied to find the optimal sequence for each shop (examples of heuristics can be found in Zhang and Van De Velde, 2012; Al-Salem, 2004). On the other hand, it has been already mentioned that it is a generalization of the PFSP, which is one of the main combinatorial optimization problems (Zhang et al., 2009) and in consequence one of the most studied scheduling problems (Pan et al., 2008a). As mentioned before, the PFSP was proved to be NP-complete for more than three machines. In order to provide efficient approximate methods for this problem, numerous contributions have been presented in the literature (see e.g. Framinan et al., 2004; Reza Hejazi and Saghafian, 2005; Ruiz and Maroto, 2005 for recent reviews). Among them, the NEH heuristic (Nawaz et al., 1983) stands out as one of the most efficient heuristics for the problem. Basically, the NEH algorithm consists of two phases:

1. Jobs are sorted according to descending sums of processing times.

2. The first job resulting from the previous phase is placed in a partial sequence. Then, each remaining job j (from $j = 2 \cdots n$) in the sorted list is inserted in all possible positions of the partial sequence, thus building j partial sequences. The partial sequence with the best (lowest) makespan is chosen as reference for the following iteration. The procedure finishes when the n jobs are sequenced.

The complexity of the NEH is defined by its second phase, which has a complexity of $O(n^3 \cdot m)$ due to the fact that, in each iteration k (with $k \in [1 \cdots n]$) the evaluation of the k partial sequences can be completed in $O(n^2 \cdot m)$. The complexity was reduced to $O(n^2 \cdot m)$ by Taillard (1990) by proposing a mechanism (named Taillard's acceleration in the following) to perform the evaluations of the k partial sequences in $O(n \cdot m)$. Additionally, different approaches have been introduced in the literature to treat the tie-breaking of the NEH algorithm outperforming the original tie-breaking mechanism (see e.g. Fernandez-Viagas and Framinan, 2014 and Kalczynski and Kamburowski, 2008). Regarding the initial order of the NEH algorithm, Framinan et al. (2003) studied different initial orders showing that the original order (decreasing sum of processing times) remained the best one.

The first heuristics to solve the DPFSP were proposed by Naderi and Ruiz (2010). They suggested four constructive heuristics, denoted NEH1, NEH2, VND(a) and VND(b), following

the ideas taken from the PFSP and employing Taillard's acceleration. More specifically, NEH1 and NEH2 are adaptations to the problem of the original NEH by means of using two assignment rules to choose the factory where a job has to be introduced were tried, i.e.:

1. To allocate the job to the factory with the lowest current makespan (included in the NEH1 heuristic).

2. To allocate the job to the factory that can process it at the earliest time (used in NEH2 algorithm).

On the other hand, VND(a) and VND(b) are composed of a first step in which the NEH2 heuristic is performed, and then its solution is improved by means of a simple variable neighborhood descent consisting of two different neighborhoods and two different acceptance criteria, one for VND(a) and another for VND(b).

Using NEH2 and VND(a) as initial solutions, Gao and Chen (2011) were the first authors who proposed an iterated optimization algorithm for the problem. More specifically, they presented a genetic algorithm with a local search phase including mechanisms of exchange and insertion of jobs. Their proposal was later outperformed by the tabu search algorithm (TS) by Gao et al. (2013), in which partial sequences of two different factories were iteratively exchanged and improved by means of several enhanced local search based also on methods of exchange and insertion of jobs. Next, Wang et al. (2013) implemented an Estimation of Distribution Algorithm (EDA), although their proposal was not compared with the tabu search algorithm from Gao et al. (2013), arguing that no results were listed for direct comparisons. Finally, Lin et al. (2013) proposed for the problem a variation of the iterated greedy, denoted MIG, in which the size of the destruction was randomly chosen between two bounds and the temperature of a simulated annealing-like acceptance criterion decreased with the iterations. However, they do not include any local search phase in their algorithm to improve the solution. Their algorithm was compared with algorithms implemented in Gao and Chen (2011); Gao et al. (2013) concluding that the MIG is the most efficient. However, the CPU times for each instance of the testbed were different for each heuristic under comparison so there is up-to-now no comparison of the state-of-the-art algorithms under the same exact conditions.

To summarise the state-of-the-art in the DPFSP problem, there are two types of algorithms available:

1. Very fast heuristics, i.e. NEH1, NEH2, VND(a) and VND(b). They are simple or composite heuristics (see notation of Framinan et al., 2005) where the computational time is non controllable by the decision-maker and the solutions can be quickly found even for large size of the problem.

2. Iterative improvement algorithms, i.e. TS, EDA, and MIG. The improvement phase of these algorithms is performed iteratively improving the solutions –usually based on any of the aforementioned very fast heuristics– at the expense of substantially increasing the computation times. To the best of our knowledge, there are no direct comparison among these algorithms under the same conditions in the literature.

In this paper, we focus in the second type of algorithms. Thereby, we first implement these algorithms (MIG, TS and EDA) using the same computer and programming language and reporting the results in Section 4. By doing so, a direct comparison among these algorithms is provided. Additionally, in Section 3, we propose a new heuristic for the DPFSP that uses the specific structure of solutions of the problem to reduce the search space and that improves the results obtained by existing algorithms, as we show in a subsequent computational experience in Section 4.

3 The proposed bounded-search iterated greedy algorithm (BSIG)

In this paper we propose an algorithm labeled Bounded-Search Iterated Greedy (BSIG) which can be seen as a special case of the Iterated Greedy (IG) algorithm (Ruiz and Stützle, 2007). The IG starts with an initial solution and then iteratively applies four steps. First, a number of jobs are taken out of the sequence. Then these jobs are again inserted in the sequence, one by one, following a greedy procedure until no more job has to be inserted. After that, a local search mechanism

is performed in order to improve the solution. Finally, a simulated annealing procedure decides if the actual sequence is kept as iteration sequence. IG is one of the best algorithms for the PFSP (see e.g. Pan et al., 2008a) and it has been successfully applied to a variety of scheduling problem, such as the sequence dependent setup times problem (Ruiz and Stützle, 2008), flowshop scheduling problem with blocking (Ribas et al., 2011), unrelated parallel machine scheduling (Fanjul-Peyro and Ruiz, 2010), or the no-wait flowshop scheduling problem (Pan et al., 2008b).

BSIG also iteratively constructs and destructs a solution trying to improve it in each iteration by means of three local search phases. In order to reduce the search space, two properties are derived in Subsection 3.1. In Subsection 3.2, the main procedure of BSIG is described. Its construction phase is detailed in Subsection 3.3, while in Subsections 3.4 and 3.5, two new local search phases are described. Finally, in Section 3.6 a full factorial design of experiment is carried out in order to obtain the best values of the parameters of the algorithm.

3.1 Problem Properties

It is clear that, when a job is assigned to a specific factory in the DPFSP, the makespan of this factory is increased by a certain value. The following properties serve us to define lower and upper bounds for this makespan:

Property 3.1 ***(Lower bound on makespan)****: The makespan of a factory with m machines must increase at least* $min_i\,(p_{i,l})\,\forall p_{i,l} > 0$ *with* $i \in [1, \cdots, m]$ *when a new job l is assigned to this factory, regardless its specific position in the schedule of this factory.*

Proof. The proof of this property is obvious: Before assigning job l in position k of the sequence, there was a set H of machines (with $|H| \geq 1$ being at least one in the first machine) where there was no idle time between positions k-1 and k (worst case). Hence, the new completion times of the jobs placed after position k are increased at least the processing time $p_{h,l}$ in each machine $h \in H$ and at least $min_{h \in H}\,(p_{h,l})$ in the rest of machines, with $min_{h \in H}\,(p_{h,l}) \geq min_{i \in [1,\cdots,m]}\,(p_{i,l})$. This fact proves that the makespan of the factory after introducing the job increases at least the minimum processing time of the job l. □

Property 3.2 ***(Upper bound of makespan)****: The makespan of a factory with $n-1$ jobs and with m machines must increase at most $\sum_{i=1}^{m} p_{i,l}$ with $i \in [1, \cdots, m]$ when a new job l is assigned to this factory, regardless its specific position in the schedule of the factory.*

Proof. The proof of the property is obvious using the same reasoning as in Property 3.1. □

Both properties can be useful for the DPFSP and have been taken into account in the design of BSIG. Since BSIG includes iterations where a job has to be assigned to one of the f factories, there are f possible options. The idea is to discard the options where its lower bound (according to Property 3.1) is higher than the current best value of the objective function. The effect in the reduction of the search space of Property 3.1 will be explained in detail in Section 4.1. However, Property 3.2 was not found to be significant for the algorithm and it has not been incorporated in the BSIG. Note that more sophisticated (and tighter) lower bounds for the makespan of each factory have been tested, however they have not resulted in a significant improvement of the objective function due to the increase in the CPU time when the lower bound is calculated. In contrast, the simple lower bound of Property 3.1 can be calculated without increasing the complexity of the algorithm, since the minimum processing time of each job is obtained at the beginning of the algorithm.

3.2 Main Procedure

The main procedure of the proposed algorithm is summarised in Figure 1. It starts with the implementation of the fast constructive heuristic NEH2 of Naderi and Ruiz (2010). Once an initial solution is so-obtained, it is improved by using three different local search methods (LS1, RLS1 and RLS2) before entering in the iterated procedure. The local search method LS1 was presented in Naderi and Ruiz (2010) and is a simple iterative improvement algorithm for each factory using a first-improvement type pivoting rule. This local search method has been successfully applied in numerous algorithms for PFSP to minimize makespan and total flowtime (see e.g. Ruiz and Stützle, 2007; Li et al., 2009). In Figure 2 the pseudo code of this method is shown.

When the solution cannot be further improved, the algorithm begins an iterative procedure composed of the following four phases, which are repeated until the stopping criteria is reached:

- Destruction phase: This phase tries to perturb the solution and –together with the simulated annealing phase– its objective is to provide diversification of solutions. In this phase, d jobs are randomly chosen to be removed of the sequence without repetition forming a new sequence denoted π_1.

- Construction phase: Each one of the aforementioned d jobs is inserted, one by one, in the sequence following a greedy procedure (`ConstructionFunction`, see subsection 3.3) using Taillard's acceleration and Property 3.1 , i.e. when a job is to be inserted, there are f factories where the job can be assigned. Property 3.1 is used to discard factories in which the insertion of the job does not improve the current makespan.

- Improvement phase: It serves to implement the intensification of the algorithm. The sequence constructed in the last phase is improved using three different local search methods: LS1, RLS1 and RLS2. The complexity of these local search methods are $n^2 \cdot m/F$, $n^2 \cdot m/F$ and $n^3 \cdot m/F^2$ respectively being the RLS2 the procedure with highest complexity. This fact may cause a non-efficient behaviour of the heuristic when n is very high in comparison with F since it could lead to a low diversification. Therefore, RLS2 is used only when n/F is lower or equal than a parameter L. By doing so, we get a total complexity of $n^2 \cdot m/F$ for the local search phase of the algorithm.

- Simulated annealing phase: A simple simulated annealing criterion is introduced in the algorithm with a constant $Temperature$ which is a function of a parameter T of the algorithm:

$$Temperature = T \cdot \frac{\sum_{\forall i} \sum_{\forall j} p_{i,j}}{n \cdot m \cdot 10}$$

3.3 ConstructionFunction

As Taillard's acceleration allows performing the insertion phase with low complexity, π_d (the destructed jobs) are inserted in the sequence using a similar procedure as in NEH2 (see `ConstructionFunction` in Figure 3). The d destructed jobs are introduced, one by one, in the position of sequence π_1 which minimises the makespan. If we denote by C_{max}^{ref} the reference makespan or best-known

```
π := NEH2(decreasing sum of processing times) ;
for f = 1 to F do
|  π^f := LS1(π^f) ;
end
flag := true;
while flag do
|  π := RLS1(π) ;
|  if solution improved then
|  |  flag := false;
|  end
end
if n/F ≤ L then
|  flag := true;
|  while flag do
|  |  π := RLS2(π) ;
|  |  if solution improved then
|  |  |  flag := false;
|  |  end
|  end
end
π_b := π;
while stopping criterion is not reach do
|  π_1 := π;
|  for i = 1 to d do
|  |  π_1 := randomly remove a job from π_1 and insert it in π_D;
|  end
|  π_2 := ConstructionFunction(π_1, π_D)
|  for f = 1 to F do
|  |  π_2^f := LS1(π_2^f) ;
|  end
|  π_3 := RLS1(π_2) ;
|  if n/F ≤ L then
|  |  π_3 := RLS2(π_3) ;
|  end
|  if C_max(π_3) < C_max(π) then
|  |  π := π_3
|  |  if C_max(π_3) < C_max(π_b) then
|  |  |  π_b := π_3
|  |  end
|  else if random ≤ exp{−(C_max(π_3) − C_max(π))/Temperature} then
|  |  π := π_3
|  end
end
return π_b
```

Figure 1: Main Procedure of the BSIG

```
flag := true;
while flag do
    flag := false;
    for f = 1 to F do
        C_max^ref = C_max(π^f);
        Remove job π^f[i] placed in position i of the factory f.
        Test job π^f[i] in any possible position of π^f (using Taillard's accelerations) and
        place it in the position with the lowest makespan
        if C_max(π^f) < C_max^ref then
            flag := true;
            break;
        end
    end
end
```

Figure 2: Local Search LS1

makespan, then it is clear that it makes sense to assign the job to a factory only if its lower bound (calculated according to Property 3.1) is lower than the reference makespan. This mechanism serves to decrease the number of factories where the jobs are tried (bounded search mechanism). The procedure of this bounded search in each iteration is relative simple: First the job π_d is tried to be placed in the first factory and the best makespan is kept as C_{max}^{ref}. Secondly, the job is tried to be assigned to factories f which satisfy $C_{max}^{f} + min_i\,(p_{i,\pi_d}) < C_{max}^{ref}$. C_{max}^{ref} is then updated when the new makespan (due to the insertion of job π_d in factory f) improves the actual C_{max}^{ref}.

3.4 Simple Relative Local Search, RLS1

The relative local search RLS1 searches for better solutions by inserting jobs from one factory to another. More specifically, each job of the factory with maximum makespan is tried to be scheduled in all positions of each factory verifying Property 3.1. If the global makespan improves, then the job is scheduled in the factory that minimises the makespan. The procedure then restarts first by looking for the new factory with maximum makespan until each job assigned to the factory with maximum makespan is tried without solution improvement. The pseudo code of RSL1 is shown in Figure 4.

```
Function ConstructionFunction(π, π_D)
    for d = 1 to D do
        Test job π_D[d] in any possible position of π^{f=1} (using Taillard's accelerations) and
        denote C^{ref}_{max} the best makespan.
        p_min := minimum processing time of job π_D[d] in any machine i;
        for f = 2 to F do
            C^f_{max} := makespan of the factory f;
            if C^f_{max} + p_min < C^{ref}_{max} then
                Test job π_D[d] in any possible position of the factory f (using Taillard's
                accelerations) and denote C^0_{max} the best makespan.
                if C^0_{max} < C^{ref}_{max} then
                    C^{ref}_{max} = C^0_{max};
                end
            end
        end
        π := permutation obtained by inserting π_D[d] in the factory and in the position
        with less makespan;
    end
    return π;
end
```

Figure 3: Procedure ConstructionFunction

3.5 Relative Local Search based in exchange, RLS2

The relative local search RLS2 exchanges jobs between factories. Thereby, each job of the factory with the maximum makespan is tried to be exchanged with each job of the rest of the factories. In order to apply Taillard's acceleration, each of both jobs is removed of the sequences of the factories and inserted in the other factory looking for the best position there. This procedure is repeated for each pair of jobs of both factories choosing the pair of jobs and positions minimizing the makespan of the chosen factories. If the new maximum makespan of both factories is lower than in the last iteration, the procedure begins again by the factory with maximum makespan. The procedure stops when each job of the factory of maximum makespan is tested without improving the solution. The pseudo code of this relative local search is shown in Figure 5.

3.6 Experimental Parameter Tuning

The proposed algorithm is composed of three parameters T, d, L. In order to find the best values of the parameters, a full factorial design of experiment is performed for the BSIG with the following

Procedure *RLS1(π)*

$h(f) = 1$, for $f = 1 \cdots F$;
$i(f) = 1$, for $f = 1 \cdots F$;
$\pi_b := \pi$ being $\pi = \left[\pi^1, \cdots, \pi^f, \cdots, \pi^F\right]$;
Determine the factory f_{max} with maximum makespan (C'_{max})
while $i(f_{max}) < n_{f_{max}}$ **do**
  $j := h(f_{max}) \bmod n_{f_{max}}$;
  $\pi_0 :=$ remove job $\pi^{f_{max}}[j]$ from $\pi^{f_{max}}$;
  $C^{f_{max}}_{max} :=$ makespan of the sequence π_0;
  $p_{min} :=$ minimum processing time of job $\pi^{f_{max}}[j]$ in any machine i;
  for $f = 1$ **to** F **do**
    $C^f_{max} :=$ makespan in the factory f;
    if $C^f_{max} + p_{min} < C'_{max}$ **then**
      Test job $\pi^{f_{max}}[j]$ in each position of the factory f (using Taillard's accelerations)
    end
  end
  $\pi :=$ permutation obtained by inserting $\pi^{f_{max}}[j]$ in the factory and in the position with less makespan;
  Determine the factory f_{max} with maximum makespan (C_{max})
  if $C_{max} < C'_{max}$ **then**
    $C'_{max} = C_{max}$;
    $i(f_{max}) = 1$;
    $\pi_b := \pi$;
  else
    $i(f_{max})++$;
  end
  $h(f_{max})++$;
end
return π_b;
end

Figure 4: Relative Local Search RLS1

Procedure *RLS2(π)*

$h(f) = 1$, for $f = 1 \cdots F$;
$i(f) = 1$, for $f = 1 \cdots F$;
$\pi_b := \pi$ being $\pi = \left[\pi^1, \cdots, \pi^f, \cdots, \pi^F\right]$;
Determine the factory f_{max} with maximum makespan (C'_{max})
while $i(f_{max}) < n_{f_{max}}$ **do**
  $C^{aux}_{max} = C'_{max}$;
  $flag :=$ false;
  $j := h(f_{max}) \bmod n_{f_{max}}$;
  for $f = 1$ **to** F **do**
    for $g = 1$ **to** n_f **do**
      if $f \neq f_{max}$ **then**
        $\pi_0 :=$ remove job $\pi^{f_{max}}[j]$ from $\pi^{f_{max}}$;
        $\pi_1 :=$ remove job $\pi^f[g]$ from π^f;
        Best makespan C^1_{max} due to testing job $\pi^{f_{max}}[j]$ in each position of π_1 denoting Pos_f the chosen position (using Taillard's acceleration).
        Best makespan C^0_{max} due to testing job $\pi^f[g]$ in each position of π_0 denoting $Pos_{f_{max}}$ the chosen position (using Taillard's acceleration).
        if $C^1_{max} < C^{aux}_{max} \& C^0_{max} < C^{aux}_{max}$ **then**
          $flag :=$ true;
          $BestPos_{f_{max}} := Pos_{f_{max}}$;
          $BestPos_f := Pos_f$;
          $chosen_g := g$;
          $chosen_f := f$;
          $C^{aux}_{max} = max\left(C^0_{max}, C^1_{max}\right)$;
        end
      end
    end
  end
  if $flag$ **then**
    $\pi^{f_{max}} :=$ permutation obtained by inserting $\pi^{chosen_f}[chosen_g]$ in the factory f_{max} and in the position $BestPos_{f_{max}}$;
    $\pi^{chosen_f} :=$ permutation obtained by inserting $\pi^{f_{max}}[j]$ in the factory $chosen_f$ and in the position $BestPos_f$;
    Update π with $\pi^{f_{max}}$ and π^{chosen_f};
    Determine the factory f_{max} with maximum makespan (C_{max})
  end
  if $C_{max} < C'_{max}$ **then**
    $C'_{max} = C_{max}$;
    $i(f_{max}) = 1$;
    $\pi_b := \pi$;
  else
    $i(f_{max})++$;
  end
  $h(f_{max})++$;
end
return π_b;
end

Figure 5: Relative Local Search with interchange RLS2

Source	Df	Chi-Square	Sig.
Parameter d	4	43.453	0.000
Parameter T	4	4.353	0.360
Parameter L	3	5.717	0.126

Table 1: Kruskal-Wallis for the parameters d, L and T

level of the parameters:

- $T \in [0.1, 0.2, 0.3, 0.4, 0.5]$
- $d \in [3, 4, 5, 6, 7]$
- $L \in [15, 20, 25, 30]$

The BSIG is evaluated by means of relative percentage deviation (RPD) which is defined as follows:

$$RPD = \frac{C_{max} - Base}{Base} \cdot 100$$

where C_{max} is the makespan obtained by the BSIG and $Base$ is the solution obtained by an alternative algorithm (VNDa).

Each combination of parameters has been tested for 96 combination of n, m and F, i.e. $n \in [20, 50, 100, 200]$, $m \in [5, 10, 15, 20]$ and $F \in [2, 3, 4, 5, 6, 7]$ using 5 instances for each one representing a total of 480 instances where the processing times of each job in each machine was uniformly distributed between 1 and 99. Note that we perform 5 runs of each instance due to the randomness of the algorithm. Each replicate is stopped when the CPU time reaches a limit of $n \cdot m \cdot F \cdot 1.5$ milliseconds. Since the normality and homoscedasticity assumptions were not confirmed, a non-parametric Kruskal-Wallis analysis is used to determine statistically difference between the parameters. A summary of the results is shown in Table 1. It can be observed that there are statistically significant differences only between the levels of parameter d, being the level of all other parameters non-statistically significant. Additionally, the analysis reveals that the best values for the parameters were found for $d = 5$, $L = 20$ and $T = 0.4$, so these are the values used in the subsequent computational experience.

4 Computational Evaluation

The proposed BSIG is compared with the best available algorithms for the DPFSP: MIG, EDA and TS. In order to define the most efficient algorithm, the same computer conditions have to be used, which implies that the heuristics have to be implemented:

- Under the same computer.
- Using the same programming language.
- And using the same stopping criterion for compared heuristics in each instance of the testbed.

Thereby, each algorithm is again implemented in C# in the same computer, a PC with 2.80 GHz Intel Core i7-930 processor and 16 GBytes of RAM memory. The algorithms are evaluated for all instances presented by Naderi and Ruiz (2010) which are available in http://soa.iti.es. A total of 720 instances are included in this testbed varying the number of jobs, machines and factories according to the following values $n \in [20, 50, 100, 200, 500]$, $m \in [5, 10, 15, 20]$ and $F \in [2, 3, 4, 5, 6, 7]$ and using 10 instances for each combination of parameters. In order to increase the power of the analysis, 5 runs have been performed per instance for each algorithm. Regarding the stopping criteria, we have used three different stopping criteria based on computation time for the heuristics: $n \cdot m \cdot F \cdot 0.5$, $n \cdot m \cdot F \cdot 1$ and $n \cdot m \cdot F \cdot 2$ milliseconds (see e.g. Ruiz and Stützle, 2007; Lin et al., 2013 for similar stopping criteria in the literature). Thereby, each heuristic has been stopped when the computation time reaches these values.

The performance of the algorithms is again evaluated by means of relative percentage deviation (RPD) which is now defined as follows:

$$RPD_i = \frac{C_{max,i} - Best}{Best} \cdot 100$$

where $C_{max,i}$ is the makespan obtained by the algorithm i and $Best$ is the best known solution taken from `http://soa.iti.es`.

The $ARPD$ values are shown in Table 2 and 3 for the stopping criterion $n \cdot m \cdot F \cdot 0.5$ milliseconds, in Tables 4 and 5 for the stopping criterion $n \cdot m \cdot F \cdot 1$ and, finally, in Tables 6 and 7

n x m	BSIG	TS	EDA	MIG
20 x 5	4.94	8.75	5.99	5.34
20 x 10	4.28	7.14	5.17	4.44
20 x 20	3.69	5.78	4.28	3.80
50 x 5	0.50	2.63	4.77	1.64
50 x 10	0.92	3.46	5.48	1.73
50 x 20	0.85	3.01	4.59	1.32
100 x 5	0.14	0.90	5.81	1.02
100 x 10	0.40	1.59	7.76	1.41
100 x 20	0.62	1.66	7.22	1.37
200 x 10	0.23	0.69	9.58	0.93
200 x 20	0.44	0.97	10.27	1.18
500 x 20	0.19	0.70	12.44	0.92
Average	1.43	3.11	6.95	2.09

Table 2: $ARPD$ (by n and m) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 0.5$ milliseconds

f	BSIG	TS	EDA	MIG
2	1.45	2.35	6.27	1.78
3	1.22	2.52	6.57	1.80
4	1.12	2.84	6.71	1.82
5	1.12	3.07	6.86	1.85
6	1.47	3.49	7.26	2.27
7	2.25	4.36	8.01	3.03
Average	1.43	3.11	6.95	2.09

Table 3: $ARPD$ (by f) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 0.5$ milliseconds.

for for $n \cdot m \cdot F \cdot 2$ milliseconds yielding e.g 1.43 for the BSIG heuristic, 3.11 for TS algorithm, 6.95 for the EDA and 2.09 for the MIG heuristic according to the first stopping criterion. The results show that the BSIG heuristic outperforms the rest of the heuristics for all stopping criteria. In fact, BSIG outperforms MIG, TS and EDA for each size of the problem regardless the stopping criterion. Additionally, the BSIG algorithm finds the best solution for 93.0% instances, while MIG, TS and EDA found the best solution for 1.9%, 3.0% and 12.6%, respectively. Comparing the results with the best known solution for the largest CPU time, in 263 of the 720 instances (36.53%) new best solutions are found by BSIG. In contrast, only 0 new best solutions were found for TS, 38 instances for MIG and 0 for EDA.

n x m	BSIG	TS	EDA	MIG
20 x 5	4.80	8.67	5.66	5.08
20 x 10	4.18	7.20	4.93	4.29
20 x 20	3.60	5.81	4.19	3.69
50 x 5	0.21	2.64	3.70	0.97
50 x 10	0.55	3.53	4.72	1.10
50 x 20	0.55	3.06	3.96	0.89
100 x 5	-0.01	0.93	4.84	0.46
100 x 10	0.11	1.56	6.83	0.72
100 x 20	0.32	1.69	6.55	0.71
200 x 10	-0.02	0.62	8.69	0.33
200 x 20	0.14	0.89	9.48	0.43
500 x 20	-0.09	0.54	11.95	0.23
Average	1.20	3.10	6.29	1.58

Table 4: $ARPD$ (by n and m) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 1$ milliseconds

f	BSIG	TS	EDA	MIG
2	1.16	2.34	5.63	1.45
3	0.95	2.50	5.83	1.34
4	0.88	2.79	6.00	1.28
5	0.87	3.04	6.23	1.31
6	1.27	3.50	6.64	1.67
7	2.05	4.40	7.43	2.41
Average	1.20	3.10	6.29	1.58

Table 5: $ARPD$ (by f) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 1$ milliseconds.

n x m	BSIG	TS	EDA	MIG
20 x 5	4.72	8.74	5.55	4.92
20 x 10	4.11	7.16	4.88	4.19
20 x 20	3.57	5.76	4.19	3.64
50 x 5	0.00	2.65	2.99	0.43
50 x 10	0.26	3.50	4.01	0.70
50 x 20	0.31	3.03	3.45	0.60
100 x 5	-0.18	0.89	3.92	0.10
100 x 10	-0.14	1.61	6.04	0.20
100 x 20	0.01	1.66	5.99	0.24
200 x 10	-0.22	0.66	7.81	-0.01
200 x 20	-0.12	0.88	8.86	-0.02
500 x 20	-0.34	0.43	11.44	-0.13
Average	1.00	3.08	5.76	1.24

Table 6: $ARPD$ (by n and m) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 2$ milliseconds

f	BSIG	TS	EDA	MIG
2	0.96	2.30	4.96	1.15
3	0.74	2.50	5.26	0.98
4	0.65	2.74	5.46	0.94
5	0.67	3.05	5.72	0.94
6	1.06	3.52	6.18	1.32
7	1.89	4.39	6.99	2.09
Average	1.00	3.08	5.76	1.24

Table 7: $ARPD$ (by f) of the heuristics BSIG, TS, MIG and EDA for the stopping criteria of $n \cdot m \cdot F \cdot 2$ milliseconds.

Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
TS vs BSIG	1.672	1.525	1.561	1.784	29.452	0.000
EDA vs BSIG	5.511	3.715	5.240	5.783	39.839	0.000
MIG vs BSIG	0.656	0.457	0.623	0.690	38.548	0.000

Table 8: Paired samples t-test for stopping criterion of $n \cdot m \cdot F \cdot 0.5$ milliseconds.

Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
TS vs BSIG	1.899	1.522	1.788	2.011	33.504	0.000
EDA vs BSIG	5.096	3.607	4.832	5.359	37.930	0.000
MIG vs BSIG	0.379	0.315	0.356	0.402	32.329	0.000

Table 9: Paired samples t-test for stopping criterion of $n \cdot m \cdot F \cdot 1$ milliseconds.

Additionally, a paired samples t-test is carried out in order to compare the heuristics for each stopping criterion. These tests can be applied since the random variables ($ARPD$) are related (the same test bed is used for each algorithm) and the hypothesis of independence can be rejected. The results of the tests (see Tables 8, 9 and 10) show that BSIG statistically improves each other algorithm being the maximum p-value 0.000.

4.1 Impact of reduction of the search space

The proposed BSIG includes a mechanism (using Property 3.1) to reduce the number of solutions to be evaluated. In this section, the impact of this mechanism on the effectiveness of the heuristic is analysed by comparing the performance of the proposed algorithm with and without the bounded search mechanism for each stopping criterion. The results are shown in Table 11 and in Table 12 aggregated by the number of factories, and by n and m, respectively. In both tables, the second column indicates the average percentage of branches (factories) discarded in the functions that use this mechanism, whereas the third column shows the average reduction in the

Algorithm	Mean	SEM	IC - Lower	IC - Upper	t	Significance
TS vs BSIG	2.084	1.500	1.974	2.194	37.306	0.000
EDA vs BSIG	4.763	3.497	4.507	5.019	36.572	0.000
MIG vs BSIG	0.239	0.247	0.221	0.258	26.077	0.000

Table 10: Paired samples t-test for stopping criterion of $n \cdot m \cdot F \cdot 2$ milliseconds.

f	Average Discarded Factories (%)	Decrease in the number of iterations (%)
2	30.71%	14.74%
3	33.05%	12.71%
4	32.99%	13.93%
5	32.48%	9.01%
6	32.31%	9.85%
7	32.33%	9.86%
Average	32.31%	11.68%

Table 11: Impact of the bounded search mechanism with the number of factories.

number of iterations in the BSIG when employing this mechanism. Additionally, the results have been calculated averaging for the three stopping criteria. Summarizing, it was obtained that a 32.31% of the branches (factories) are discarded in the construction phase and in the RLS1 of the proposed iterated algorithm. Note that there is a substantial decrease of the discarded factories with the increase in the number of machines of the problem for the same number of jobs. This is due to the fact that the chosen lower bound is $min_i\,(p_{i,l})$ with $i \in [1, \cdots, m]$ and it therefore less tight as m increases. Furthermore, the number of iterations of the proposed BSIG for each analysed stopping criterion is increased an 11.7% in average. The difference between this value for large and small size instances is due to RLS2, which does not include the bounded search and needs large computational time when used. Both the number of discarded factories and the decrease in the number of iterations stress the importance of the bounded search mechanism in the algorithm.

5 Conclusions

The Distributed Permutation Flowshop Scheduling Problem (DPFSP) consists of two interrelated decision problems: First, jobs are assigned to be processed in one of the f identical factories of the company. Secondly, the sequence of jobs in each factory is determined taking into account that each job has the same manufacturing flow through each one of the m machines. To solve the problem, we have presented a new algorithm (BSIG) consisting of an iterative destruction and greedy construction of the solution with three local search phases. BSIG employs a property of the problem to estimate the makespan of a factory when a new job is inserted, so the search space can

n x m	Average Discarded Factories (%)	Decrease in number of iteration (%)
20 x 5	40.42%	8.50%
20 x 10	25.73%	5.07%
20 x 20	19.54%	4.04%
50 x 5	45.79%	6.13%
50 x 10	30.94%	4.79%
50 x 20	19.30%	3.16%
100 x 5	51.90%	21.11%
100 x 10	35.80%	12.65%
100 x 20	21.64%	8.58%
200 x 10	40.27%	27.47%
200 x 20	25.18%	14.86%
500 x 20	31.22%	23.84%
Average	32.31%	11.68%

Table 12: Impact of the bounded search mechanism order with the problem size n and m.

be reduced. The evaluation of the performance of BSIG was compared with that of the existing algoritms TS, EDA and MIG using the instances presented by Naderi and Ruiz (2010) for three different stopping criteria. Each algorithm was implemented under the same computer conditions. Furthermore, paired samples t-tests were carried out to determine statistically differences between the heuristics. The comparison shows that the proposed BSIG outperforms existing heuristics, thus being the most efficient iterative improvement algorithm for the problem (with a p value of 0.000). On the one hand, comparing the four heuristics, the best solution of the four heuristics was found by BSIG 2008 times out of a total of 2160 instances (summarising results of the three stopping criteria). On the other hand, using the proposed heuristic, a new best known solution was found in 263 of the 720 instances (36.5%) using the stopping criterion of $n{\cdot}m{\cdot}F{\cdot}2$ milliseconds. Additionally, the effect of the bounded search method in the algorithm was analysed reporting a decrease in the computational times of 32.31% whenever applied.

Acknowledgements

This research has been funded by the Spanish Ministry of Science and Innovation, under the project "SCORE" with reference DPI2010-15573/DPI.

References

Al-Salem, A. (2004). A heuristic to minimize makespan in proportional parallel flow shops. *International Journal of Computing & Information Sciences*, 2(2):98–107.

Cao, D. and Chen, M. (2003). Parallel flowshop scheduling using tabu search. *International Journal of Production Research*, 41(13):3059–3073.

Chan, F., Chung, S., and Chan, P. (2005). An adaptive genetic algorithm with dominated genes for distributed scheduling problems. *Expert Systems with Applications*, 29(2):364–371.

Chan, H. and Chan, F. (2010). Comparative study of adaptability and flexibility in distributed manufacturing supply chains. *Decision Support Systems*, 48(2):331–341.

Chung, S., Chan, F., and Chan, H. (2009). A modified genetic algorithm approach for scheduling of perfect maintenance in distributed production scheduling. *Engineering Applications of Artificial Intelligence*, 22(7):1005–1014.

Fanjul-Peyro, L. and Ruiz, R. (2010). Iterated greedy local search methods for unrelated parallel machine scheduling. *European Journal of Operational Research*, 207(1):55–69.

Fernandez-Viagas, V. and Framinan, J. M. (2014). On insertion tie-breaking rules in heuristics for the permutation flowshop scheduling problem. *Computers and Operations Research*, 45(0):60 – 67.

Framinan, J., Gupta, J., and Leisten, R. (2004). A review and classification of heuristics for permutation flow-shop scheduling with makespan objective. *Journal of the Operational Research Society*, 55(12):1243–1255.

Framinan, J., Leisten, R., and Rajendran, C. (2003). Different initial sequences for the heuristic of nawaz, enscore and ham to minimize makespan, idletime or flowtime in the static permutation flowshop sequencing problem. *International Journal of Production Research*, 41(1):121–148.

Framinan, J., Leisten, R., and Ruiz-Usano, R. (2005). Comparison of heuristics for flowtime minimisation in permutation flowshops. *Computers and Operations Research*, 32(5):1237–1254.

Gao, J. and Chen, R. (2011). A hybrid genetic algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Computational Intelligence Systems*, 4(4):497–508.

Gao, J., Chen, R., and Deng, W. (2013). An efficient tabu search algorithm for the distributed permutation flowshop scheduling problem. *International Journal of Production Research*, 51(3):641–651.

Garey, M., Johnson, D., and Sethi, R. (1976). Complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129.

Graham, R. L., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. (1979). Optimization and Approximation in Deterministic Sequencing and Scheduling: A Survey. *Annals of Discrete Mathematics*, 5:287–326.

Hatami, S., Ruiz, R., and Andrés-Romano, C. (2013). The distributed assembly permutation flowshop scheduling problem. *International Journal of Production Research*.

Jia, H., Fuh, J., Nee, A., and Zhang, Y. (2007). Integration of genetic algorithm and gantt chart for job shop scheduling in distributed manufacturing systems. *Computers and Industrial Engineering*, 53(2):313–320.

Jia, H., Nee, A., Fuh, J., and Zhang, Y. (2003). A modified genetic algorithm for distributed

scheduling problems. *Journal of Intelligent Manufacturing*, 14(3-4):351–362.

Johnson, S. (1954). Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1(1):61–68.

Kahn, K., Castellion, G., and Griffin, A. (2004). *The PDMA Handbook of New Product Development: Second Edition*. Wiley.

Kalczynski, P. J. and Kamburowski, J. (2008). An improved NEH heuristic to minimize makespan in permutation flow shops. *Computers & Operations Research*, 35(9):3001–3008.

Li, X., Wang, Q., and Wu, C. (2009). Efficient composite heuristics for total flowtime minimization in permutation flow shops. *Omega*, 37(1):155–164.

Lin, S.-W., Ying, K.-C., and Huang, C.-Y. (2013). Minimising makespan in distributed permutation flowshops using a modified iterated greedy algorithm. *International Journal of Production Research*, 51(16):5029–5038.

Moon, C., Kim, J., and Hur, S. (2002). Integrated process planning and scheduling with minimizing total tardiness in multi-plants supply chain. *Computers and Industrial Engineering*, 43(1-2):331–349.

Naderi, B. and Ruiz, R. (2010). The distributed permutation flowshop scheduling problem. *Computers and Operations Research*, 37(4):754–768.

Nawaz, M., Enscore, Jr, E. E., and Ham, I. (1983). A Heuristic Algorithm for the m-Machine, n-Job Flow-shop Sequencing Problem. *OMEGA, The International Journal of Management Science*, 11(1):91–95.

Pan, Q.-K., Tasgetiren, M., and Liang, Y.-C. (2008a). A discrete differential evolution algorithm for the permutation flowshop scheduling problem. *Computers and Industrial Engineering*, 55(4):795–816.

Pan, Q.-K., Wang, L., and Zhao, B.-H. (2008b). An improved iterated greedy algorithm for the no-wait flow shop scheduling problem with makespan criterion. *International Journal of Advanced Manufacturing Technology*, 38(7-8):778–786.

Reza Hejazi, S. and Saghafian, S. (2005). Flowshop-scheduling problems with makespan criterion: A review. *International Journal of Production Research*, 43(14):2895–2929.

Ribas, I., Companys, R., and Tort-Martorell, X. (2011). An iterated greedy algorithm for the flowshop scheduling problem with blocking. *Omega*, 39(3):293–301.

Ruiz, R. and Maroto, C. (2005). A comprehensive review and evaluation of permutation flowshop heuristics. *European Journal of Operational Research*, 165(2):479–494.

Ruiz, R. and Stützle, T. (2007). A simple and effective iterated greedy algorithm for the permutation flowshop scheduling problem. *European Journal of Operational Research*, 177(3):2033–2049.

Ruiz, R. and Stützle, T. (2008). An iterated greedy heuristic for the sequence dependent setup times flowshop problem with makespan and weighted tardiness objectives. *European Journal of Operational Research*, 187(3):1143–1159.

Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. *European Journal of Operational Research*, 47(1):65–74.

Vairaktarakis, G. and Elhafsi, M. (2000). The use of flowlines to simplify routing complexity in two-stage flowshops. *IIE Transactions (Institute of Industrial Engineers)*, 32(8):687–699.

Wang, S.-Y., Wang, L., Liu, M., and Xu, Y. (2013). An effective estimation of distribution algorithm for solving the distributed permutation flow-shop scheduling problem. *International Journal of Production Economics*, 145(1):387–396.

Zhang, X. and Van De Velde, S. (2012). Approximation algorithms for the parallel flow shop problem. *European Journal of Operational Research*, 216(3):544–552.

Zhang, Y., Li, X., and Wang, Q. (2009). Hybrid genetic algorithm for permutation flowshop scheduling problems with total flowtime minimization. *European Journal of Operational Research*, 196(3):869–876.